KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X
11/2011

KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

Sie können die elektronische Form des KaffeeKlatschs
monatlich, kostenlos und unverbindlich
durch eine E-Mail an

abo@bookware.de

abonnieren.

Ihre E-Mail-Adresse wird ausschließlich für den Versand
des KaffeeKlatschs verwendet.

Editorial

Inkompetenzkompensationskompetenz

In einer Geburtstagsrede für einen Berufskollegen hat der Philosoph Odo Marquard 1973 den Begriff der Inkompetenzkompensationskompetenz geprägt. Er wollte damit auf den Missstand der Philosophie hinweisen und darauf, wie sie in diese missliche Lage geraten ist. Sie sei einmal „kompetent für alles“ gewesen, wäre heute aber „kompetent nur noch für eines: nämlich für das Eingeständnis der eigenen Inkompetenz“ [1].

Nicht, dass ich der Meinung wäre, dass auch die Software-Entwicklung am Ende wäre. Ganz im Gegenteil. Aber die schier nicht zu bewältigende Masse an Möglichkeiten lässt zwangsläufig alle Software-Entwickler immer inkompetenter werden, was aber – dass mich ja niemand falsch versteht – zunächst einmal völlig wertfrei ist.

Viel interessanter ist nun, dass gerade diese Inkompetenz in unserem Bereich zu ungeahntem Ideenreichtum und unglaublichen Leistungen führt. Mit der uns zur Verfügung stehenden Kompensationskompetenz können wir nämlich nicht nur alle Aufgaben lösen, sondern darüber hinaus auch sämtliche Probleme mehr oder weniger souverän meistern. Wer würde sich also deswegen über die Zunahme der Inkompetenz beklagen wollen.

Unsere Fähigkeiten – oder besser noch die nicht vorhandenen Fähigkeiten – sind ja eigentlich immer ein „Problem“, weil wir ja grundsätzlich unsere Arbeit nur mit dem erledigen (können), was wir gelernt haben. Nicht umsonst heißt es doch bei uns, dass man „in jeder Programmiersprache ein gutes Fortran-Programm schreiben kann“. Es ist ja auch gar nicht so einfach, alte Gewohnheiten abzulegen. Jeder, der beispielsweise den Wechsel von strukturierter zur objektorientierter Programmierung mitgemacht hat, weiß wovon ich rede.

Unabhängig davon lässt man uns all zu oft gar keine Zeit, um uns überhaupt mit den Neuerungen und Verbesserungen vertraut zu machen. Und wer könnte sich losgelöst davon auch schon den Luxus leisten, tatsächlich jeden Versionswechsel mitzumachen. Denn der würde auch nichts nutzen, weil die für das Projekt relevanten Fehler tendenziell nicht behoben werden; und wenn doch, dann müssen meist clevere Workarounds wieder rückgängig gemacht werden.

Allerdings schaffen wir es auch, ungeeignete Hilfsmittel so unglaublich zweckzuentfremden, dass sie plötzlich auch für Domänen geeignet zu sein scheinen, an die der Werkzeugmacher nicht im entferntesten gedacht hat. So bemüht man sich z. B. redlich darum, mit Hilfe von kaskadierenden Methoden- und Funktionsaufrufen in beliebigen Programmiersprachen einfache, vermeintlich besser lesbare englische „Sätze“ zu schreiben. Eine „Interne DSL“ nennt man das dann, und nimmt billigend in Kauf, dass ein solches Vorgehen die Laufzeit empfindlich verschlechtert.

Ein bisschen ist es wie eine Krankheit: Sobald wir uns irgendwo nicht zurechtfinden, etwas gar nicht erst finden können oder uns mit etwas nicht abfinden wollen, dann wird das, was nicht passt, einfach passend gemacht oder sogar neu geschrieben. Denn scheinbar fällt damit unsere Inkompetenz nicht so sehr auf.

Aber gerade weil wir so unzufrieden mit dem sind, was uns zur Verfügung steht, und weil wir Aufgaben erledigen müssen, die all zu häufig nicht so recht zu diesen Mitteln passen, und weil wir viel zu oft viel zu wenig Einfluss auf die Wahl der Werkzeuge haben, müssen wir ja auch so viel kompensieren – mit unglaublich viel Kompetenz.

Ihr Michael Wiedeking
Herausgeber

Referenzen

[1] Wikipedia *Inkompetenzkompensationskompetenz*, http://de.wikipedia.org/wiki/Inkompetenzkompensationskompetenz

Beitragsinformation

Der KaffeeKlatsch dient Entwicklern, Architekten, Projektleitern und Entscheidern als Kommunikationsplattform. Er soll neben dem Know-how-Transfer von Technologien (insbesondere Java und .NET) auch auf einfache Weise die Publikation von Projekt- und Erfahrungsberichten ermöglichen.

Beiträge

Um einen Beitrag im KaffeeKlatsch veröffentlichen zu können, müssen Sie prüfen, ob Ihr Beitrag den folgenden Mindestanforderungen genügt:

- Ist das Thema von Interesse für Entwickler, Architekten, Projektleiter oder Entscheider, speziell wenn sich diese mit der Java- oder .NET-Technologie beschäftigen?
- Ist der Artikel für diese Zielgruppe bei der Arbeit mit Java oder .NET relevant oder hilfreich?
- Genügt die Arbeit den üblichen professionellen Standards für Artikel in Bezug auf Sprache und Erscheinungsbild?

Wenn Sie uns einen solchen Artikel, um ihn in diesem Medium zu veröffentlichen, zukommen lassen, dann übertragen Sie Bookware unwiderruflich das nicht exklusive, weltweit geltende Recht

- diesen Artikel bei Annahme durch die Redaktion im KaffeeKlatsch zu veröffentlichen
- diesen Artikel nach Belieben in elektronischer oder gedruckter Form zu verbreiten
- diesen Artikel in der Bookware-Bibliothek zu veröffentlichen
- den Nutzern zu erlauben diesen Artikel für nicht-kommerzielle Zwecke, insbesondere für Weiterbildung und Forschung, zu kopieren und zu verteilen.

Wir möchten deshalb keine Artikel veröffentlichen, die bereits in anderen Print- oder Online-Medien veröffentlicht worden sind.

Selbstverständlich bleibt das Copyright auch bei Ihnen und Bookware wird jede Anfrage für eine kommerzielle Nutzung direkt an Sie weiterleiten.

Die Beiträge sollten in elektronischer Form via E-Mail an redaktion@bookware.de geschickt werden.

Auf Wunsch stellen wir dem Autor seinen Artikel als unveränderlichen PDF-Nachdruck in der kanonischen KaffeeKlatsch-Form zur Verfügung, für den er ein unwiderrufliches, nicht-exklusives Nutzungsrecht erhält.

Leserbriefe

Leserbriefe werden nur dann akzeptiert, wenn sie mit vollständigem Namen, Anschrift und E-Mail-Adresse versehen sind. Die Redaktion behält sich vor, Leserbriefe – auch gekürzt – zu veröffentlichen, wenn dem nicht explizit widersprochen wurde.

Sobald ein Leserbrief (oder auch Artikel) als direkte Kritik zu einem bereits veröffentlichten Beitrag aufgefasst werden kann, behält sich die Redaktion vor, die Veröffentlichung jener Beiträge zu verzögern, so dass der Kritisierte die Möglichkeit hat, auf die Kritik in der selben Ausgabe zu reagieren.

Leserbriefe schicken Sie bitte an leserbrief@bookware.de. Für Fragen und Wünsche zu Nachdrucken, Kopien von Berichten oder Referenzen wenden Sie sich bitte direkt an die Autoren.

Werbung ist Information

Firmen haben die Möglichkeit Werbung im KaffeeKlatsch unterzubringen. Der Werbeteil ist in drei Teile gegliedert:

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

Die Werbeflächen werden als Vielfaches von Sechsteln und Vierteln einer DIN-A4-Seite zur Verfügung gestellt.

Der Werbeplatz kann bei Frau Natalia Wilhelm via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-16 gebucht werden.

Abonnement

Der KaffeeKlatsch erscheint zur Zeit monatlich. Die jeweils aktuelle Version wird nur via E-Mail als PDF-Dokument versandt. Sie können den KaffeeKlatsch via E-Mail an abo@bookware.de oder über das Internet unter www.bookware.de/abo bestellen. Selbstverständlich können Sie das Abo jederzeit und ohne Angabe von Gründen sowohl via E-Mail als auch übers Internet kündigen.

Ältere Versionen können einfach über das Internet als Download unter www.bookware.de/archiv bezogen werden.

Auf Wunsch schicken wir Ihnen auch ein gedrucktes Exemplar. Da es sich dabei um einzelne Exemplare handelt, erkundigen Sie sich bitte wegen der Preise und Versandkosten bei Alexandra Specht via E-Mail unter alexandra.specht@bookware.de oder telefonisch unter 09131/8903-14.

Copyright

Impressum

KaffeeKlatsch Jahrgang 4, Nummer 11, November 2011
ISSN 1865-682X
Bookware – eine Initiative der MATHEMA Software GmbH

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Michael Wiedeking
Anzeigen: Natalia Wilhelm
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Kolumnen

CoffeeScript

von Andreas Schubert

CoffeeScript ist eine kleine Programmiersprache, die in letzter Zeit einige Aufmerksamkeit auf sich zog, nicht zuletzt durch ihre Aufnahme in die aktuelle *Ruby-on-Rails*-Version. In diesem Artikel wird die Sprache anhand einiger Beispiele kurz vorgestellt.

Frühlingserwachen

von Thomas Haug

Enterprise-Anwendungen zu entwerfen und zu implementieren ist eine schwierige und umfangreiche Aufgabe. Das Open-Source-Framework *Spring.Net* erleichtert die Erstellung von solchen Enterprise-Anwendungen. Hierzu stellt das Framework einen Komponenten-Container zur Verfügung, in dem Komponenten instanziiert und mittels *Dependency Injection* miteinander verdrahtet werden können. Weiterhin bietet das *Spring.Net*-Framework Abstraktionen, um gängige Frameworks wie *NHibernate* vereinfacht nutzen zu können.

Expansionistisch

von Michael Wiedeking

Heute kann man sich das zwar immer weniger vorstellen, aber damals wie heute ist es gelegentlich nötig, Informationen zu komprimieren und in wenigen Bits unterzubringen. Dann stellt sich aber die Frage, wie man diese Daten mit so wenig Aufwand wie möglich wiederherstellen kann.

CoffeeScript

It's just JavaScript

von Andreas Schubert

CoffeeScript ist eine kleine Programmiersprache, die in letzter Zeit einige Aufmerksamkeit auf sich zog, nicht zuletzt durch ihre Aufnahme in die aktuelle *Ruby-on-Rails*-Version. In diesem Artikel wird die Sprache anhand einiger Beispiele kurz vorgestellt.

Die schönen Seiten aus JavaScript

Die Programmiersprache *JavaScript* erlebt zur Zeit eine Art „zweiten Frühling", da sie mehr und mehr als ernst zu nehmende Sprache wahrgenommen wird. Vorbei sind die Zeiten, da JavaScript bestenfalls für nervige Pop-Ups oder blinkende Web-Seiten herhalten musste.

Dank immer besser werdender Bibliotheken wie zum Beispiel *jQuery* [1], der plattformübergreifenden Entwicklung für mobile Endgeräte [2] oder selbst der Server-seitigen Programmierung etwa mit *Node.js* [3] sowie immer schneller werdenden JavaScript-Laufzeitumgebungen wird die Sprache immer häufiger und in immer größeren Projekten eingesetzt. Weiterhin ist JavaScript eine Programmiersprache, welche auf nahezu jedem Betriebssystem, das einen aktuellen Browser hat, zu finden ist.

Einen schlechten Ruf hat JavaScript allerdings bei vielen Entwicklern immer noch, was vielleicht auch an der Syntax liegt. Aber hinter den vielen Klammern und Strichpunkten hat JavaScript ein tolles Objektmodell.

CoffeeScript ist nun der Versuch, nur das Beste von JavaScript in eine „schöne" Syntax zu packen.

„It's just JavaScript"

Das ist die goldene Regel, der CoffeeScript [4] folgt. Die Sprache wir durch einen Compiler in JavaScript übersetzt, sodass zur Laufzeit keine weitere Interpretation des Codes stattfindet.[1] Dieser Umstand ist auch einer der größten Schwachpunkte von CoffeeScript und soll gleich ganz am Anfang erwähnt werden.

Die Fehlersuche ist schwierig

Fehlermeldungen zur Laufzeit beziehen sich natürlich auf den compilierten JavaScript-Code und nicht auf den Quellcode in CoffeeScript. Zwar ist es nicht all zu schwer, aus der Zeile im JavaScript-Code auf die Zeile in CoffeeScript zu schließen, optimal ist das jedoch nicht. Es gibt jedoch Bestrebungen, dies zumindest im Firefox zu ändern ([5] und [6]).

Installation

Je nachdem, auf welchem Betriebssystem CoffeeScript installiert werden soll, ist unterschiedlich viel zu tun. Auf jeden Fall wird *Node.js* und der *Node Package Manager* – kurz *npm* – benötigt. Dann ist die Installation eigentlich nur noch ein Einzeiler:

```
npm install -g coffee-script
```

Nach diesem Schritt ist das Binary *coffee* installiert, mit dem CoffeeScript in JavaScript compiliert wird. Zu den zahlreichen Argumenten, die *coffee* bietet, gehören auch solche, die *coffee* ein Verzeichnis oder bestimmte Dateien überwachen lässt und bei Änderungen automatisch einen neuen Compiler-Vorgang durchführt.

Die Syntax der Sprache

CoffeeScript vereint Sprachelemente aus *Python*, *Lisp* und *Ruby* (und natürlich JavaScript). Strichpunkte werden nicht benötigt, Blöcke werden wie in Python durch Einrücken festgelegt, Klammern für Argumente bei Funktionen können weggelassen werden.

Funktionen

In CoffeeScript wird eine Funktion durch eine optionale Liste an Parametern, einen Pfeil und den Rumpf der Funktion definiert. Die leere Funktion ist somit definiert als `->`. Hier einige kleine Beispiele:

[1] Außer ggf. dem von JavaScript.

```
square = (x) -> x * x
cube = (x) -> square(x) * x
fill = (container, liquid = "coffee") ->
  "Filling the #{container} with #{liquid}..."
```

Das letzte Beispiel zeigt, dass Parameter mit *Default*-Werten belegt werden können sowie die Variablensubstitution in Zeichenketten per *#{}*. Grundsätzlich müssen Funktionen in CoffeScript keine expliziten Rückgabewerte liefern, es wird – wie z. B. auch in Ruby – der letzte Wert implizit zurückgegeben.

Objekte und Arrays

Objekte in CoffeeScript können durch Einrückung definiert werden, in einer Syntax die der *Yet Another Markup Language (YAML)* sehr ähnelt, Kommas bei mehrzeiligen *Array*-Definitionen sind optional.

```
bitlist = [
  0,1,0
  1,0,1
  0,1,0
]

kids =
  brother:
    name: "Max"
    age: 11
  sister:
    name : "Ida"
    age: 9
```

Gültigkeitsbereich von Variablen

Der CoffeScript-Compiler kümmert sich darum, dass Variablen immer korrekt im *Lexical Scope* definiert sind. *var* muss man nie selbst schreiben.

Aus folgendem CoffeeScript-Code

```
outer = 1
changeNumbers = ->
  inner = -1
  outer = 10
inner = changeNumbers
```

wird durch den Compiler dieser JavaScript-Code erzeugt:

```
var changeNumbers, inner, outer;
outer = 1;
changeNumbers = function() {
  var inner;
  inner = -1;
  return outer = 10;
};
inner = changeNumbers();
```

Die Variablen-Definitionen sind an die höchste Stelle des nächstliegendsten *Scopes* gerutscht. *outer* wird in der Funktion nicht erneut definiert, da sie bereits im *Scope* ist. *inner* innerhalb der Funktion *changeNumbers* soll dagegen nicht in der Lage sein, die gleichnamige Variable außerhalb der Funktion zu manipulieren und wird daher in der Funktion selbst deklariert.

If/Else/Unless

In CoffeeScript werden Bedingungen ohne Klammern geschrieben. Ebenso ist die aus Ruby bekannte, nachgestellte Bedingung als Einzeiler erlaubt:

```
mood = greatlyImproved if singing

if happy and knowing
  singLittleSong()
else
  showIt()
```

Loops und Comprehensions

In CoffeeScript werden die meisten Schleifen als *Comprehensions* implementiert. Eine Comprehension ist dabei ein Ausdruck, der zurückgegeben oder zugewiesen werden kann. Ebenso sind Bedingungen möglich:

```
shortNames = (name for name in list when name.length < 5)
```

Auch das Iterieren über *Key-Value*-Paare eines Objektes lässt sich mit einer Comprehension sehr einfach lösen:

```
yearsOld = max: 10, ida: 9, tim: 11

ages = for child, age of yearsOld
  "#{child} is #{age}"
```

Klassen, Vererbung und Super

In JavaScript gibt es keine Klassen, Objektorientierung funktioniert hier über veränderbare Objekte und prototypische Delegation. Dieses Verfahren unterscheidet sich von den meisten anderen objektorientierten Programmiersprachen, was die Verwendung im Alltag erschwert. In CoffeeScript gibt es eine neue *class*-Struktur, welche die Möglichkeit schafft, Klassen einen Namen zu geben, die Elternklasse zu bestimmen, einen Konstruktor und prototypische Eigenschaften zu definieren. Auch der in JavaScript etwas heikle Gebrauch von *super* vereinfacht sich dadurch deutlich.

```
class Animal
  constructor: (@name) ->

  move: (meters) ->
    alert @name + " moved #{meters}m."
```

```
class Snake extends Animal
  move: ->
    alert "Slithering..."
    super 5

class Horse extends Animal
  move: ->
    alert "Galloping..."
    super 45

sam = new Snake "Sammy the Python"
tom = new Horse "Tommy the Palomino"

sam.move()
tom.move()
```

Ausblick und Fazit

In diesem kurzen Artikel wurde ein erster Einblick in die noch relativ junge Sprache CoffeeScript gegeben. Viele Themen wie *Splats*, *Ranges*, *Switch*, *Try/Catch* etc. wurden nicht erwähnt.

Auf der Web-Seite von CoffeeScript [4] gibt es die Möglichkeit, CoffeeScript direkt im Browser zu testen und zu sehen, welcher JavaScript-Code dabei generiert wird und diesen auch ausführen zu lassen. Die Hürde, einfach ein paar Zeilen Code auszuprobieren, ist somit äußerst gering.

Ob und inwieweit sich CoffeeScript durchsetzen wird, hängt sicher nicht zuletzt an der Frage, wie stark die Ruby-on-Rails-Community auf CoffeeScript setzt. Unterstützung für die Sprache ist in der aktuellen Version 3.1 von Rails von Haus aus enthalten.

Referenzen

[1] jQuery *jQuery: The Write less, Do More JavaScript Library*
http://jquery.com

[2] Phonegap
http://www.phonegap.com

[3] Node.js
http://nodejs.org

[4] CoffeeScript
http://jashkenas.github.com/coffee-script

[5] jashkenas/coffeescript *#558: line number mapping for debug*
https://github.com/jashkenas/coffee-script/issues/558

[6] Bugzilla@Mozilla
Bug 618650 - map JS source coordinates to source language that was translated to JS
https://bugzilla.mozilla.org/show_bug.cgi?id=618650

Kurzbiographie

Andreas Schubert ist Diplom Sozialpädagoge und als Systemadministrator, Trainer und Consultant für die MATHEMA Software GmbH tätig. Neben seinem Interesse für unixoide Betriebssysteme und deren Administration beschäftigt er sich mit der Programmiersprache Ruby und dem darauf basierenden Web-Framework Ruby on Rails.

Frühlingserwachen

Anwendungsentwicklung mit dem Spring.Net-Framework, Teil 1

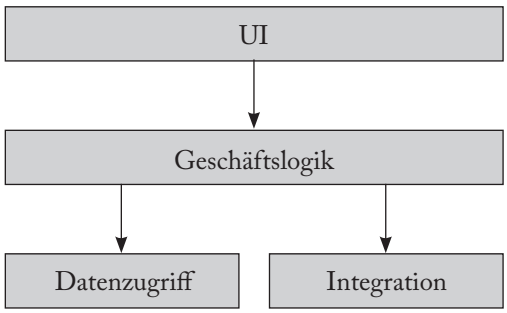
von Thomas Haug

Enterprise-Anwendungen zu entwerfen und zu implementieren ist eine schwierige und umfangreiche Aufgabe. Das Open-Source-Framework *Spring.Net* erleichtert die Erstellung von solchen Enterprise-Anwendungen. Hierzu stellt das Framework einen Komponenten-Container zur Verfügung, in dem Komponenten instanziiert und mittels *Dependency Injection* miteinander verdrahtet werden können. Weiterhin bietet das Spring.Net-Framework Abstraktionen, um gängige Frameworks wie *NHibernate* vereinfacht nutzen zu können.

In einer kleinen Artikelserie sollen wesentliche Bestandteile einer Enterprise-Anwendungsentwicklung mittels Spring.Net [1] abgedeckt werden. Als Grundlage wird im vorliegenden Artikel das Basis-Framework vorgestellt. Im nachfolgenden Artikel wird der Zugriff auf Datenbanken erläutert und die Anbindung von Drittsystemen skizziert. Im abschließenden Artikel wird dann die Erstellung von Web-Anwendungen demonstriert.

Rahmenbedingungen

In der Artikelserie soll eine Anwendung zur Verwaltung von Ersatzteilen implementiert werden. Die Anwendung soll eine typische Mehrschicht-Architektur realisieren:



Über eine (Web-basierte) Oberfläche (UI – User Interface) wird auf eine in einem Server installierte Anwendung (Geschäftslogik) zugegriffen. Diese nutzt zur Verwaltung ihrer Daten eine Datendank, die über eine Datenzugriffsschicht gekapselt ist. Fremdsysteme werden über eine Integrationsschicht angesprochen.

Neben der Entscheidung wie die Software strukturiert wird, müssen wir entscheiden, welche Frameworks und Bibliotheken eingesetzt werden. Darüber hinaus müssen auch Entschlüsse gefasst werden, wie der Lebenszyklus der einzelnen Bestandteile (sprich Komponenten) des Systems gesteuert wird und wie die Komponenten miteinander verschaltet werden können.

Diese Fragen bestimmen häufig darüber, wie gut sich das System in kommenden Iterationen erweitern und verändern lässt. Aus diesem Grund haben sich in den vergangenen Jahren verschiedene Container (z. B. *Enterprise JavaBeans*) etabliert, die feste Vorgaben zum Lebenszyklus und dem Auffinden von Komponenten machen.

Eine mittlerweile vorherrschende Art Geschäftskomponenten miteinander zu verdrahten, ist die sogenannte *Dependency Injection*, in der die Komponenten von aussen miteinander verbunden werden. Hierdurch wird die Notwendigkeit einander zu kennen auf ein mögliches Minimum reduziert. Üblicherweise beschränkt sich dieses auf reine Schnittstellenabhängigkeiten. Mehr Informationen hierzu finden sich unter [2].

Spring.Net ist eine Portierung des wohlbekannten *Dependency Injection*-Containers *Spring* aus der Java-Welt auf die .Net-Plattform. Hierbei wurden die wesentlichen Grundzüge des Frameworks beibehalten, d. h. Abstraktionen und Programmiermodell sind bis auf marginale Unterschiede identisch, sodass Spring-Programmierer mit minimalem Aufwand den Wechsel auf die .Net-Plattform vollziehen können.

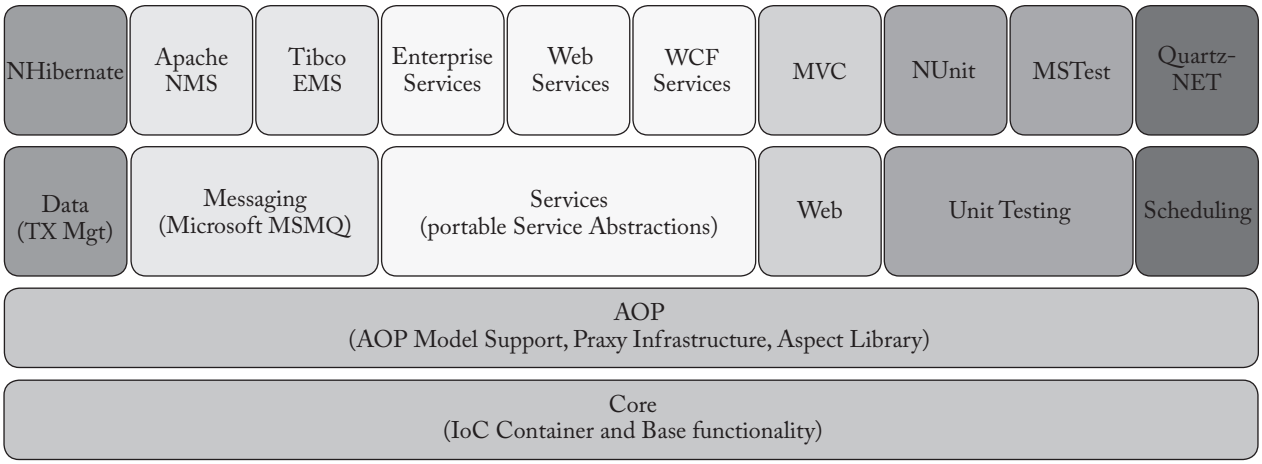


Abbildung 2: Bausteine von Spring.Net aus [3]

Überblick

Spring.Net liegt seit August 2011 in der Version 1.3.2 vor und besteht aus einer Vielzahl von Bausteinen, die den gesamten Lieferumfang des Frameworks ausmachen (siehe Abbildung 2).

Spring.Core ist die Basis des gesamten Frameworks und stellt den *Inversion of Control* (*IoC*) bereit, der mittels Dependency Injection das Verdrahten von Komponenten ermöglicht. Desweiteren bietet Spring.Core ein *Data Binding*-Framework, ein Validierungs-Framework, Unterstützung für *Threading* und eine *Expression Language* (EL) zur Navigation von Objekt-Graphen.

Spring.AOP liefert ein Framwork zur aspekt-orientierten Programmierung, um Querschnittsthemen wie *Logging*, Transaktionshandhabung, etc. in eine Enterprise-Anwendung einweben zu können, ohne die eigentliche Geschäftslogik hiermit verschmutzen zu müssen. Weiterhin ist in diesem Bestandteil auch ein *Proxy*-Framework enthalten, das für das Bereitstellen von Aspekten nötig ist.

Auf Basis der beiden genannten und grundlegenden Bestandteile bauen sich die nachfolgenden Bausteine des Gesamt-Frameworks auf:

- *Spring.Data* und *Spring.NHibernate* erleichtern das Implementieren von Persistenz mit *ADO.Net* bzw. *NHibernate* als *Objekt-Relationaler Mapper (ORM)*. Zusätzlich wird ein Transaktions-Framework angeboten, das sowohl programmatisches als auch deklaratives Transaktionsmanagement ermöglicht. Dies wird im zweiten Artikel genauer beleuchtet.
- Nachrichten-basierte Kommunikation wird mit drei *Sub*-Frameworks unterstützt: *Spring.Messaging* zur Integration von *Microsoft Message Queue (MSMQ)*, *Spring.Messaging.EMS* zur Integration des *IBM TIBCO Message Brokers* und *Spring.Messaging.NMS* für die Anbindung an *Apache Active MQ*. Die letzte Variante wird in einem weiteren Artikel aufgezeigt.
- Zugriff auf weitere für Enterprise-Anwendungen wichtige Frameworks bietet das *Sub*-System *Spring.Services*, das beispielsweise den Umgang mit *.Net Remoting*, *Enterprise Services* und *ASMX WebServices* vereinfacht.
- Auch die Entwicklung von Web-Anwendungen wird durch Spring.Net unterstützt. Hierzu bietet *Spring.Web* ein Framework auf Basis von *ASP.NET* an, das sich zum Ziel gesetzt hat, den Abstraktionsgrad anzuheben. Hierbei wird sowohl die Entwicklung nach *ASP.Net Model-View-Controler (MVC) 2* als auch *MVC 3* unterstützt.
- Um Testaufgaben für *IoC*-Anwendungen zu erleichtern, bietet Spring.Net verschiedene Pakete an um die Test-Frameworks *NUnit* und *MS Test* nahtlos in die Spring.Net-Plattform zu integrieren.

Komponenten erzeugen

Spring-Komponenten sind erst einmal gewöhnliche *POCOs (Plain Old CLR Object)*, d. h. sie müssen nicht von speziellen Schnittstellen oder (abstrakten) Klassen abgeleitet sein. Dies ist ein wesentliches Ziel des Spring.Net-Frameworks, nämlich nicht-invasiv zu sein. Die Komponenten werden in einem Container zum Leben erweckt.

Im Falle der Ersatzteil-Verwaltung schreiben wir uns eine einfache *Bootstrap*-Klasse:

```
namespace BugShopSpring
{
  class BootStrapper
  {
    static void Main(string[] args)
    {
      string url = @"Config\Services.xml";
      ApplicationContext cxt =
        new XmlApplicationContext(url);
    }
  }
}
```

Wir erzeugen ein Objekt, das die *ApplicationContext*-Schnittstelle implementiert. Hierzu nutzen wir die *XmlApplicationContext*-Klasse aus dem *Spring.Context.Support-Namespace*. Diese Klasse erlaubt es Komponentendefinitionen in XML-Dateien abzulegen und liest diese bei der Objekterzeugung per Konstruktor ein. Eine rudimentäre Komponentendefinition könnte dergestalt sein:

```
<?xml version="1.0" encoding="utf-8" ?>
<objects xmlns="http://www.springframework.net">
  <object id="ErsatzteilService"
    type="BugShopSpring.Services.Ersatzteile,
      BugShopSpring">
    <property name="Automarke" value="VW" />
  </object>
</objects>
```

In dieser Definition wird eine Komponente mit dem Schlüssel (*id*) *Ersatzteilservice* vom Typ *BugShopSpring.Services.Ersatzteile* aus der Assembly *BugShopSpring* beschrieben. Nun könnten wir den Container nach der entsprechenden Komponente „fragen“, entweder per

```
ErsatzteilService service=
  ctx["ErsatzteilService"] as ErsatzteilService;
```

oder mittels

```
ErsatzteilService service =
  ctx.GetObject("ErsatzteilService")
    as ErsatzteilService;
```

Damit Spring.Net die Komponente erzeugen kann, muss sie einen *Default-Konstruktor* anbieten. Sollte das Objekt vor der Verwendung noch konfiguriert werden müssen, so können skalare Werte über *property*-Attribute in das Objekt injiziert werden. Hierbei wird der *Setter* der entsprechenden Eigenschaft gesetzt. Dieser kann auch *private* sein.

Bietet die Komponente keinen Default-Konstruktor, so ist mit Spring.Net auch eine Konstruktor-basierte Dependency Injection möglich:

```
<object id="ErsatzteilService"
      type="BugShopSpring.Services.Ersatzteile,
            BugShopSpring">
    <constructor-arg type="string" value="VW"/>
 </object>
```

Wie wird aber der Lebenszyklus der Komponente gesteuert? Spring.Net bietet hierfür zwei Varianten an: *Singleton*- und *Prototyp*-Komponenten. Standard ist die Singleton-Komponente, hier garantiert das Spring.Net-Framework, das jeder Zugriff auf die Komponente über den *ApplicationContext* immer die gleiche Instanz zurückliefert. Bei Prototyp-Komponenten liefert jeder Aufruf über den *ApplicationContext* eine neue Instanz. Gesteuert wird diese Eigenschaft über das *singleton*-Attribut in der XML-Beschreibung der Komponente:

```
<object id="ErsatzteilService"
      type="BugShopSpring.Services.Ersatzteile,
            BugShopSpring"
    singleton= "true" | "false"
 </object>
```

Im Web-Umfeld werden allerdings weitere Lebenszyklus-Modelle angeboten. Diese werden über das Attribut *scope* eingestellt, funktionieren aber lediglich im ASP.Net-Container.

Bei der Injektion von Prototyp-Komponenten in Singleton-Komponenten müssen wir beachten, dass dies immer nur zur Erzeugungszeit der Singleton-Komponente durchgeführt wird, d. h. genau einmal. Soll die Singleton-Instanz allerdings bei jedem Aufruf eine andere Prototyp-Komponente nutzten, ist die normale Dependency Injection nicht ausreichend. In diesem Fall können wir auf *Method Injection* zurückgreifen.

Ist die Erzeugung einer Singleton-Komponente kostspielig, so können wir den Container anweisen, die Erzeugung solange herauszuzögern, bis diese benötigt wird. Durch Angabe des Attributes *lazy-init* wird dies bewerkstelligt.

In unserer Anwendung soll der Ersatzteilservice die Möglichkeit bieten, nach Ersatzteilen anhand einer Artikelnummer zu suchen. Hierzu soll der Service eine Datenzugriffskomponente nutzen, die per Dependency Injection dem Service mitgeteilt werden soll. Da wir uns bisher noch nicht mit der Persistenz auseinanderge-

setzt haben, implementieren wir eine sehr rudimentäre Lösung, die Ersatzteile im einem *Dictionary Objekt* ablegt (ohne jegliche Synchronisation, Fehlerhandhabung ect.):

```
public class ErsatzteilDAO : IErsatzteilDAO{
  private IDictionary<string, Ersatzteil> liste =
    new Dictionary<string, Ersatzteil>();

  public void Persist(Ersatzteil ersatzteil) {
    liste.Add(ersatzteil.ArtikelNummer, ersatzteil);
  }
  public Ersatzteil FindByArtNr
  (string artikelNummer) {
    return liste[artikelNummer];
  }
}
```

Damit alle Service-Komponenteninstanzen die gleiche Sicht auf die Ersatzteile haben, muss also die Datenzugriffskomponente eine Singleton sein. Da wir unsere Anwendung nach Schichten aufteilen, soll dies auch für die Spring.Net-Konfigurationen gelten. Deshalb legen wir die Konfiguration der *ErsatzteilDAO*-Komponente in eine eigene Konfiguration *DataAccess.xml*:

```
<objects xmlns="http://www.springframework.net">
  <object id="ErsatzteilDAO"
    type="BugShopSpring
      .DataAccess.MemoryPersistence.ErsatzteilDAO,
      BugShopSpring" />
</objects>
```

Nun müssen wir noch die Datenzugriffs-Komponente in unseren Service injizieren, dazu passen wir dessen Konfiguration folgendermaßen an:

```
<object id="ErsatzteilService"
  type="BugShopSpring.Services.ErsatzteilService,
    BugShopSpring"  singleton="false">
  <constructor-arg type="string" value="VW"/>
  <property name="ErsatzteilDao"
    ref="ErsatzteilDAO"/>
</object>
```

Die Property hat statt einem *Value*-Attribut ein *ref*-Attribut, dessen Inhalt die ID der entsprechenden Komponente darstellt.

Auch unsere *Bootstrapper*-Klasse müssen wir minimal anpassen, damit die neue Konfiguration auch geladen wird:

```
IApplicationContext ctx = new
XmlApplicationContext(
  @"Config\Services.xml", @"Config\DataAccess.xml"
);
```

Testen

Wir könnten nun wie bereits gezeigt unsere Service-Komponente über den Context erhalten und an diesem Geschäftsmethoden aufrufen. Allerdings wollen wir ja eine Server-Anwendung schreiben, die später von unseren Clients aufgerufen wird, d. h. in der *Bootstrapper*-Klasse sollten solche Methodenaufrufe eigentlich nicht passieren. Aus diesem Grund belassen wir erstmal unsere *Bootsprapper*-Klasse so wie sie ist und implementieren stattdessen einen *Unit*-Test, der die Funktionalität des Services überprüft.

Hierzu nutzen wir die Möglichkeiten des Spring.Net-Frameworks, *NUnit*-Tests mit Dependency Injection umzusetzen. Der zu erstellende *Unit*-Test muss sich hierfür von der Klasse *AbstractDependencyInjectionSpringContextTests* ableiten und die abstrakte Eigenschaft *ConfigLocations* überschreiben. Dieses Property liefert dem implizit gestarteten Container eine Liste von Komponentenkonfigurationen, d. h. unser Test sollte dergestalt sein:

```
namespace BugShopSpringTest.Services {
  [TestFixture]
  public class ErsatzteilServiceTest :
    AbstractDependencyInjectionSpringContextTests {
    private ErsatzteilService ErsatzteilService
      { get; set;}
      protected override string[] ConfigLocations {
      get {
        return new String[]{
          "file://Config/Services.xml",
          "file://Config/DataAccess.xml" };
      }
    }
  }
}
```

Die Eigenschaft *ErsatzteilService* wird durch den Container mittels Dependency Injection gesetzt und die Abhängigkeiten des Services werden auch durch den Container befriedigt.

Wollen wir nun eine Testmethode implementieren, so bewegen wir uns wieder im bekannten *NUnit*-Terrain:

```
[Test]
public void CheckErsatzteilservice() {
   Ersatzteil ersatzteil =
      ErsatzteilService.FindErsatzteil("12345");
   Assert.NotNull(ersatzteil,
   "Ersatzteil wurde nicht gefunden");
}
```

Die Möglichkeiten, die sich durch diesen Ansatz bieten, sind offensichtlich. Sollten wir in der Zukunft die Datenzugriffskomponente durch eine vollwertige Variante austauschen, so wird sich die Konfiguration in der Applikation ändern, aber unser *Unit*-Test kann sehr wohl noch die einfache Speicher-basierte Variante nutzen. Wir müssen lediglich verschiedene Container-Konfigurationen in die Applikation, respektive in die Testklasse, laden.

Ausblick

Wir haben in diesem Artikel den Grundstein gelegt, um unsere Enterprise-Anwendung zu implementieren. Hierzu haben wir Grundzüge des Spring.Net-Containers kennengelernt und zwei Komponenten implementiert, die mittels Dependency Injection verdrahtet wurden. Abschliessend wurde gezeigt, wie *Unit*-Testen mit dem Spring.Net-Framework erleichtert wird.

Im nachfolgenden Artikel werden wir die Persistenz der Geschäftsdaten angehen und in diesem Zuge auf NHibernate setzen. Die Transaktionshandhabung werden wir hierbei sowohl programmatisch als auch deklarativ beschreiben. Hierdurch werden wir die eigentlichen Stärken des Spring.Net-Frameworks nutzen.

Ressourcen:

[1] Spring.Net *Spring.Net - Application Framework*, Version 1.3.2, http://www.springframework.net

[2] Martin Fowler *Inversion of Control Containers and the Dependency Injection pattern*, http://martinfowler.com/articles/injection.html

[3] Spring.Net *home*, Reference Manual and API, Version 1.3.2, http://www.springframework.net/documentation.html#Reference_Manual_and_API

Kurzbiographie

Thomas Haug (thomas.haug@mathema.de) ist als Senior-Consultant und Trainer für die MATHEMA Software GmbH tätig. Seine Themenschwerpunkte umfassen die Java Standard und Enterprise Edition (Java SE und Java EE) sowie das .NET-Umfeld, insbesondere im Hinblick auf Enterprise-Anwendungen. Neben seiner Projekttätigkeit hält er Technologietrainings für MATHEMA und berät verschiedene Projekte hinsichtlich des optimalen Einsatzes von .NET- oder Java-Technologien.

Des Programmierers kleine Vergnügen

Expansionistisch

von Michael Wiedeking

Heute kann man sich das zwar immer weniger vorstellen, aber damals wie heute ist es gelegentlich nötig, Informationen zu komprimieren und in wenigen Bits unterzubringen. Dann stellt sich aber die Frage, wie man diese Daten mit so wenig Aufwand wie möglich wiederherstellen kann.

Will man beispielsweise Farbe komprimieren und dafür nur 16 Bit verbrauchen, so kann man zum Beispiel für den Rot-, Grün- und Blauanteil der Farbe je fünf Bit stiften (einer der Anteile darf sogar sechs Bit haben). Hat man nun eine Grafikkarte, die acht Bit pro Farbanteil braucht, stellt sich die Frage, wie man die fünf Bit am geschicktesten auf die nötigen acht Bit aufteilt. Schwarz soll dabei natürlich schwarz bleiben und weiß möglichst weiß.

Der naheliegende Ansatz ist eine entsprechende Multiplikation. Mit fünf Bit lässt sich der Bereich von 0 bis 31 abdecken, mit acht Bit der von 0 bis 255. Eine Multiplikation mit 255/31 ≈ 8,22 erfüllt dann den entsprechenden Zweck. 0 bleibt dann 0 und die 31 wird – so man korrekt rundet – auf 255 abgebildet. Aber man ahnt es schon, eine Multiplikation mit einer Gleitkommazahl nebst Umwandlung in die verschiedenen Zahlformate ist aus Gründen der Performanz nicht akzeptabel.

Fünf Bit, acht Bit, da scheint doch ein Verschieben der Bits angebracht. Dazu müssen doch die Bits einfach nur um drei Bit-Positionen verschoben werden. 0 bleibt dann 0 und die 31 wird auf 248 abgebildet. Allgemein lässt sich also dieses Problem, von der vorhandenen Bit-Breite n auf die neue Breite m zu kommen, dadurch lösen, indem man einfach den Ausgangswert v um $m - n$ Stellen nach links verschiebt.

Allerdings stellt sich die Frage, ob es nicht noch besser geht, denn schließlich werden durch das Verschieben nur Nullen ergänzt. Wie kann man nun erreichen, dass diese neuen $m - n$ Bits einen vernünftigeren Wert bekommen? Indem man anstatt der Nullen die höherwertigen Bits zum Auffüllen verwendet. Denn diese sind fast Null, wenn der Wert klein ist, und bestehen aus vielen Bits, wenn der Wert groß ist.

```
static int expand(int v, int n, int m) {
  int result = v << (m – n);
  return result | (result >> n);
}
```

Damit ist der maximale Wert in obigem Beispiel nicht mehr 248 sondern korrekt 255. Sobald aber die Zielgröße m größer wird als $2n$, verschlechtert sich natürlich das Ergebnis mit der Anzahl der „fehlenden“ Bits. Sollen beispielsweise die fünf Bit auf zwölf Bit expandiert werden, dann sind die niederwertigsten zwei Bits immer Null und das Maximum entsprechend statt 4096 nur noch 4092.

Diese Variante liefert also wie gesagt nur dann korrekte Werte, wenn $m \leq 2n$ ist, in allen anderen Fällen muss man doch auf die Multiplikationsvariante zurückgreifen. Ob in diesem Fall eine Reduktion auf eine ganzzahlige Multiplikation immer noch einen akzeptablen Maximalwert liefert, muss genau so entschieden werden, wie beim Maximalwert mit „fehlenden“ Bits. Sonst erhält man statt des strahlenden Weiß vielleicht nur ein ganz trübes Grau.

Kurzbiographie

Michael Wiedeking (michael.wiedeking@mathema.de) ist Gründer und Geschäftsführer der MATHEMA Software GmbH, die sich von Anfang an mit Objekttechnologien und dem professionellen Einsatz von Java einen Namen gemacht hat. Er ist Java-Programmierer der ersten Stunde, „sammelt“ Programmiersprachen und beschäftigt sich mit deren Design und Implementierung.

Deutsch für Informatiker

Wortschatz

von Alexandra Specht

Um ehrlich zu sein[1] fällt mir langsam nichts Hübsches mehr ein, mit dem ich Sie, verehrte Leserinnen und Leser, noch beglücken kann. Ich will weder Sie noch mich mit irgendeiner abstrusen Grammatik langweilen, noch will ich wieder einmal beliebte Fehler und deren Lösungen auflisten. Wer zum Beispiel *dass* und *das* immernoch verwechselt, kann in einer alten Ausgabe nachsehen. Das sehe ich nämlich tatsächlich so häufig, dass ich selber schon immer überlege, wie es denn eigentlich richtig ist und ob nicht doch irgendwie irgendwann beides immer richtig sein darf, weil es einfach so oft falsch gemacht wird, dass es schon absurd ist. Aber analog zu dem alten juristischen Spruch *Kein Recht im Unrecht* [1] sollte man doch eventuell berücksichtigen, dass das *das,* das das Verständnis des Inhaltes eines Satzes oft erheblich erleichtert, dann so und nur das *dass,* das Sätze aneinander reiht, mit zwei *s* zu schreiben ist. Aber gut jetzt.

Ich versuche mir natürlich auch Anregungen für diese Kolumne zu holen, indem ich andere Kolumnen, Artikel, Bücher und Essays lese. Mein Problem daran ist aber leider, dass ich diese nicht wirklich als Anregung für eigene Kolumnen nehmen kann, weil ich tatsächlich immer dazu neige, die Sprache und den Ausdruck der Autorin oder des Autors anzunehmen. Nicht schön. Und ich möchte auch ungern 20.000 Euro zahlen müssen wie Karl-Theodor zu Guttenberg.

Darum beschränke ich mich hier und heute darauf, ein Buch eines Schreibers vorzustellen, der deutlich kreativer ist als ich. Sascha Lobo ist Autor der Kolumne S.P.O.N. bei Spiegel Online [2] und hat gerade in einem Buch *Wortschatz* [3] Wörter zusammengetragen, die alle Neuschöpfungen sind. Grundlage des Buches ist eine Kolumne der Zeitschrift Neon, die sich mit neuen Wörtern beschäftigt, die teilweise auch von Lesern vorgeschlagen werden.

Ich möchte Ihnen ein paar dieser Wörter vorstellen, damit Sie sich selber ein Bild machen können, ob das Buch neue Wörter enthält, die in Ihren Sprachschatz aufgenommen werden sollten.

Das Buch ist in viele Kapitel aufgeteilt, da weiß man dann gleich, wo man nachschauen muss, wenn man in einer besonderen Lebenslage ein treffendes Wort finden möchte. Die Kapitel sind unter anderem:

Arbeit & Büro, *Familie & Freunde*, *Liebe & Sex*, *Biologisches & Biounlogisches*, *Neue grammatische Entwicklungen*, *Neue emotionale und rationale Zustände*, *Digitale Welt*, *Neue Worte für neue negative Gefühle*, *Neue Euphemismen*, *Worte für unterwegs*, *Schattenworte*.

[1] Da fällt mir ein, dass ich einmal gelesen habe, dass immer dann, wenn jemand sagt *Um ehrlich zu sein…* eine Lüge kommt. Aber ich meine es leider wirklich ehrlich.

Hier ein paar Wörter aus dem Kapitel *Arbeit & Büro*:

Nobbing

Unter *Nobbing* leidet man, wenn man im Bekanntenkreis als Einziger keine zermürbende Mobbing-Geschichte mit brüchiger Stimme vortragen kann, weil man noch nie gemobbt wurde. *Nobbing*-Betroffene berichten häufig von einem unbestimmten Gefühl, irgendwie ausgeschlossen zu sein; erste Fälle von Berufsunfähigkeit wegen *Nobbing* werden derzeit vor Gerichten verhandelt.

Feiertragisch

Wie es sich anfühlt, an einem Tag zu arbeiten, der in praktisch allen anderen Bundesländern ein Feiertag ist.

Teletic [auch: *Handyzucken*]

Das nervöse Zucken zum eigenen Telefon, sobald irgendwo ein Handy klingelt. Gegen den *Teletic* hilft es erwiesenermaßen nicht, sein eigenes Gerät auf einen absonderlichen Klingelton einzustellen, da das Gehirn seit den 90er-Jahren neuronal auf die gleichen zehn Klingeltöne konditioniert ist. Bei Patienten im fortgeschrittenen Stadium des *Teletics* sind bereits Handyzuckungen beobachtet worden, wenn im Radio klingeltonähnliche Melodiefolgen zu hören sind. Der *Teletic* ist nur durch vollständige Taubheit heilbar und kann in Verbindung mit einem klingeltonähnlichen Tinnitus direkt in den Wahnsinn führen.

Kantoffel

Bei der *Kantoffel* handelt es sich um einen entfernten Verwandten der Kartoffel, der bereits bei der Ernte extrem mehlig und vollkommen geschmacksfrei ist und deshalb in den Kantinen der Welt bevorzugt eingesetzt wird. Die Verwendung von *Kantoffeln* ist nach dem immer noch gültigen Kölner Kantoffelkonkordat von 1673 ansonsten nur in der Schweinezucht und in Universitätsmensen erlaubt.

Excem

Allergische Reaktion auf Excel. Excel ist unterdessen gesundheitsamtlich als einziges Virus anerkannt, das vom Computer auf den Menschen übertragen wird und in manchen Firmen einen Verbreitungsgrad von 104 % (bei Einberechnung freier Mitarbeiter) erreicht.

Schmeeting

Kurzform für *Scheißmeeting*. Aus jedem Meeting kann binnen Sekunden ein *Schmeeting* werden, in der Geschichte des Kapitalismus ist aber noch niemals aus einem *Schmeeting* wieder ein Meeting geworden.

Egolf

Oberhalb des mittleren Managements dehnt sich der Ich-Wettstreit vom Büro auf den Golfplatz aus. Nicht zum Spaß, sondern als Zeichen ihrer Elitenzugehörigkeit und für das eigene Ego spielen Führungskräfte *Egolf*. Das ausgeklügelte Regelwerk dieses Ich-Sports wurde bisher nicht schriftlich niedergelegt: die 15.000 Mitglieder des zuständigen *Egolf*-Verbands konnten sich bei 15.000 internen Bewerbungen noch nicht auf einen Vorstand einigen.

Kaffeerienmachen

Bei manchen Kollegen dauert die Kaffeepause eben etwas länger.

Wörter aus dem Kapitel *Toptrends & neue Buzzwords*:

Ultratasking

Der Nachfolger des Multitasking umfasst nicht nur die Fähigkeit, zwei oder mehr Handlungen zur selben Zeit auszuführen, sondern auch den gleichzeitigen, multiplen Wechsel zwischen einer Vielzahl von parallel ausgeführten Tätigkeiten in schneller Taktung. Multitasking besteht aus mehreren parallel verlaufenden Handlungen, *Ultratasking* ist ein chaotisch geflochtener, mäan- dernder Tätigkeitszopf, der in sich verzwirbelt ist. Beispiele für *Ultratasking* sind nächtliches Kampfjetfliegen im Nebel kopfüber unter Feindbeschuss, einarmiges Jonglieren mit brennenden Kettensägen auf einem Einrad im Löwenkäfig oder eine Autofahrt mit Kindern.

Mobile Cloud Evolution

Mobile Cloud Evolution ist ein bedeutungsloser Begriff aus einer Begriffsreihe, die nach einer uralten Übereinkunft von Beratern und Technikjournalisten benutzt wird, wenn sie nach dem neuesten Trend gefragt werden. Seit 1957 tritt alle zwei Jahre im Verborgenen der Welttrendrat zusammen und bestimmt mit einem komplizierten Würfelverfahren die drei neuen zusammenhanglosen Begriffe, die für die Folgezeit zum weltweiten Großtrend erklärt werden. Über geheime Mailinglisten, auf längst vergessenen Chatplattformen und in geschlossenen Facebook-Gruppen werden die aktuellen Begriffe unter den Eingeweihten verbreitet. *Mobile Cloud Evolution* gilt

noch bis zum 31.12.2013 als weltweiter Trend. Unter den Vorgängern waren *Processual Change Management* [2002], *Social Media Monitoring* [2006] und *Sell Everything Now* [2008]. Das letztgenannte Begriffstriplett ergab sich aus einem Übermittlungsfehler, wurde aber trotzdem als Großtrend verbreitet und führte durch Amokverkäufe sämtlicher Investitionsanlagen zur Finanzkrise 2008.

Hier noch ein Wort aus dem Kapitel *Neue Krankheiten*:

Allephobie

Die *Allephobie* ist nicht die Angst vor allen. *Allephobie* ist die Angst davor, dass der Handy-, Laptop- oder Kamera-Akku leer sein könnte, bevor man eine Ladechance hat. Während die meisten Phobien eher psychischen Ursprungs sind, hängt die *Allephobie* mit dem Material zusammen und ist daher mit zunehmendem Alter der Akkus absolut begründet. Ein mitgeführter Ersatzakku hat sich nicht als hilfreich erwiesen, weil dieser die Intensität der *Allephobie* verdoppelt. Schließlich können nun zwei Akkus zur Unzeit den Geist aufgeben.

Ein Wort für *Unterwegs*:

Jetlägerig

Das fehlende Adjektiv zu Jetlag.

Ein Wort aus *Digitale Welt*:

Offlining

Die vernetzte Gesellschaft ist immer häufiger und länger online – so lange, bis alle Leute ständig im Netz sind. Dieser Zustand hält einige Zeit an. Gerätschaften, Arbeitsabläufe und Sozialleben verankern sich in der ständigen Vernetzung. Damit wird es zum seltenen Luxusgut, kein Internet zu haben – der Trend zum *Offlining* setzt ein. Die Avantgarde beginnt mit der Veranstaltung von *Offline*-Partys im Untergrund. In Szenevierteln eröffnen die ersten *Kein-Internet*-Cafés, eine radikal internetfeindliche Splitterpartei, die Trockenpiraten, gründet sich. *Offline* wird offiziell verboten. Sektenähnliche *Offline*-Kammern, rundumhüllt mit dichtmaschigen Metallgittern, finden sich versteckt in den größeren Städten. Sogar fahrbare *Offline*-Kammern gibt es, getarnt als Blutspendebus oder als Truck für dümmliche Roadshows. *Offlining* gilt unterdessen als terrorismusnahe Tat – wer offline ist oder sein möchte, hat etwas zu verbergen. Zur Terrorismusbekämpfung und Veronlinung der Weltbevölkerung wird schließlich bei jedem Kind kurz vor der Geburt in die vordere Hirnschlagader ein Nanochip eingepflanzt, dessen Entfernung automatisch zum Tod führt. Wenige Jahre später stirbt das *Offlining* aus, und alle sind online, immer.

Und zum Schluß noch zwei Wörter aus *Familie und Freunde*:

Erzug

Sehr harte Erziehung. Erziehung verhält sich zu *Erzug* wie eine kiffende Waldorflehrerin zu Margot Honecker.

Exenverbrennung

Die *Exenverbrennung* erfolgt kurz nach dem Moment, an dem anderthalb bis drei Wochen nach dem endgültigen Ende der Beziehung der Liebeskummer in Wut umschlägt. Eben täuschte man sich selbst noch vor, gar nicht mehr so genau zu wissen, wo genau die Habseligkeiten des Ex-Partners in der Wohnung herumlägen. Schon rennt man herum, sammelt Liebesbriefe, Erinnerungsfotos und andere Beweise der einstigen Zusammengehörigkeit, wirft sie in die Badewanne und zündet sie an. Nach der *Exenverbrennung* reinigt man sich traditionellerweise durch rituelles, 24-stündiges Weinen. Anschließend lügt man sich und den Freunden vor, dass alles wieder gut sei.

So. Ich denke, Sie können sich ein Bild machen, welche Art von Wörtern in dem Buch zu finden sein werden. Immerhin 698. Das sind ca. 650 Ideen mehr, als ich bisher für die Kolumne hatte. Was ich mit dieser machen werde, bildet sich für mich noch nicht ab; irgendwelche Inspirationen?

Referenzen

[1] Wikipedia *Gleichbehandlung im Unrecht*
http://de.wikipedia.org/wiki/Gleichbehandlung_im_Unrecht

[2] SPIEGELONLINE *S.P.O.N. – Die Mensch-Maschine*
http://www.spiegel.de/thema/spon_lobo

[3] Sascha Lobo *Wortschatz: 698 neue Worte für alle Lebenslagen*, rororo 2011

Kaffeesatz

Kleine Welt

von Michael Wiedeking

Die Welt ist ein Dorf. Warum das so ist, kann ich nicht genau sagen, aber es gibt Indizien dafür. Immer wieder hört man von Menschen, die irgendwelche Bekannten an den unmöglichsten Stellen auf der Welt zufällig treffen und jeder hat schon einmal davon gehört, dass jeder jeden im Schnitt über nur sechs Ecken kennt. Und jetzt – so scheint es – ist die Welt noch einmal etwas kleiner geworden.

Wer kennt sie nicht, die Annahme, dass jeder auf der Welt jeden um ungefähr sechs Ecken kennt. Die Vermutung wurde schon 1967 vom Psychologen Stanley Milgram erhärtet, der in einem Experiment Briefe von Teilnehmern in Omaha und Wichita in den USA an eine bestimmte Adresse in Boston schicken lies. Dabei sollten die Teilnehmer den Brief nur an jemanden weitergeben, den sie persönlich kannten und von dem sie glaubten, dass er den Empfänger mit höherer Wahrscheinlichkeit kannte.

Die jeweiligen Empfänger mussten dann eine Postkarte an Milgrams Institut schicken, damit dort der Weg nachvollzogen werden konnte. Und das Experiment förderte Erstaunliches zu Tage: Die durchschnittliche Pfadlänge betrug nämlich nur 5,5. Von den 60 Briefen kamen zwar nur drei an, aber die Wissenschaftler schlossen trotzdem daraus, dass jeder amerikanische Bürger nur durch sechs andere von jedem anderen getrennt ist [1].

Die Idee der „kleinen“ sozial vernetzten Welt ist übrigens schon viel älter, denn schon 1929 schrieb der ungarische Autor Frigyes Karinthy in einem seiner Geschichten darüber und lässt einen Erzähler sagen, ihm sei eine beliebige Person der damals 1,5 Milliarden Einwohner der Erde zu zeigen, und er würde wetten, dass er mit dieser über eine Kette von höchstens fünf Personen verbunden ist, wenn jeder von diesen dazu jeweils nur auf ihre persönlichen Bekannten zurückgreift [2].

Neben Milgram, der das Experiment einige Jahre später in Varianten wiederholte und zu vergleichbaren Ergebnissen kam, publizierte der Analytiker Casper Goffman 1969 einen Artikel über die Beziehungen von Koautoren zu dem ungarischen Mathematiker Paul Erdős – der sogenannten *Erdős*-Zahl. Warum ausgerechnet Erdős als Zentrum dieses Netzwerks gewählt wurde, wird klar, wenn man weiß, dass er – schon zu Lebzeiten eine Legende – mit 1525 Publikationen mehr veröffentlicht hat, als jeder andere Mathematiker.[1]

Erdős selbst wird in diesem Zusammenhang die Zahl 0 zugewiesen, seinen Koautoren die Zahl 1, deren Koautoren die Zahl 2 et cetera und alle ohne jede Verbindung den Wert Unendlich. Auch hier ist erstaunlich, dass dieser Wert entweder unendlich oder sehr klein ist. Der Durchschnittswert der *Erdős*-Zahlen der knapp 270 000 Personen mit einem endlichen Wert liegt bei nur 4,65 [4].

Über die Jahre sind noch viele Untersuchungen dieser Art gemacht worden. Dabei hat man sich als moderne Variante des Originalexperiments der E-Mails bedient; mehrere Web-Sites sind ins Leben gerufen worden, bei denen man Verknüpfungen erstellen kann; und diverse Apps – auch unter Facebook – konnten die Verbindung zwischen zwei Personen anzeigen. Letzteres zeigte beispielsweise eine durchschnittliche Distanz von 5,73 bei einem Maximum von 12.

Neueste Studien aber zeigen, dass die Welt tatsächlich noch kleiner ist [5]. Bei dem neusten Facebook-Experiment mit der ungeheuren Teilnehmerzahl von 721 Millionen (sic!) ergab sich eine durchschnittliche Distanz von nur 4,74! Die Amerikaner unter sich sind übrigens noch besser verknüpft: Angeblich sind die Hälfte aller US-Bürger über 13 Jahre bei Facebook und die kennen sich alle über nur noch 4,37 Ecken.

Referenzen

[1] Wikipedia *Kleine-Welt-Phänomen*, http://de.wikipedia.org/wiki/Kleine-Welt-Ph%C3%A4nomen

[2] Wikipedia *Six degrees of separation*, http://en.wikipedia.org/wiki/Six_Degrees_of_Separation

[3] Wikipedia *Paul Erdős*, http://en.wikipedia.org/wiki/Paul_Erd%C5%91s

[4] Wikipedia *Erdős-Zahl*, http://de.wikipedia.org/wiki/Erd%C5%91s-Zahl

[5] Markoff, John; Sengupta, Somini *Separating You and Me? 4.74 Degrees*, www.nytimes.com/2011/11/22/technology/between-you-and-me-4-74-degrees.html

Kurzbiographie

Michael Wiedeking (michael.wiedeking@mathema.de) ist Gründer und Geschäftsführer der MATHEMA Software GmbH, die sich von Anfang an mit Objekttechnologien und dem professionellen Einsatz von Java einen Namen gemacht hat. Er ist Java-Programmierer der ersten Stunde, „sammelt“ Programmiersprachen und beschäftigt sich mit deren Design und Implementierung.

[1] Einzig Leonhard Euler kann es mit ihm aufnehmen, der zwar nicht mehr Werke, aber dafür mehr Seiten publiziert hat [3].

User Groups

Fehlt eine User Group? Sind Kontaktdaten falsch? Dann geben Sie uns doch bitte Bescheid.

Bookware
Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0
Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de

Java User Groups

Deutschland

JUG Berlin Brandenburg
http://www.jug-bb.de
Kontakt: Herr Ralph Bergmann (orga@jug-bb.de)

Java User Group Saxony
Java User Group Dresden
http://www.jugsaxouy.de
Kontakt: Herr Torsten Rentsch (torsten@jugsaxony.de)
Herr Falk Hartmann (falk@jugsaxony.de)
Herr Kristian Rink (kristian@jugsaxony.de)

rheinjug e.V.
Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf
http://www.rheinjug.de
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
Universitätsstr. 12, 45127 Essen
http://www.ruhrjug.de
Kontakt: Herr Heiko Sippel (heiko.sippel@ruhrjug.de)

JUGF
Java User Group Frankfurt
http://www.jugf.de
Kontakt: Herr Alexander Culum
(javausergroupfrankfurt@googlemail.com)

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o asc-Dienstleistungs GmbH
Ehrengard-Schramm-Weg 11, 37085 Göttingen
http://www.java.de (office@java.de)

JUG Hamburg
Java User Group Hamburg
http://www.jughh.org

JUG Karlsruhe
Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
Am Fasanengarten 4, 76131 Karlsruhe
http://jug-karlsruhe.de
jug-karlsruhe@gmail.com

JUGC
Java User Group Köln
http://www.jugcologne.org
Kontakt: Herr Michael Hüttermann
(michael@huettermann.net)

jugm
Java User Group München
Jupiterweg 8, 85586 Poing
http://www.jugm.de
Kontakt: Herr Andreas Haug (ah@jugm.de)

JUG Münster
Java User Group für Münster und das Münsterland
http://www.jug-muenster.de
Kontakt: Herr Thomas Kruse (tkjugi@sforce.org)

JUG MeNue
Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
http://www.jug-n.de
Kontakt: Frau Alexandra Specht
(alexandra.specht@jug-n.de)

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
Siekstraße 4, 38444 Wolfsburg
http://www.jug-ostfalen.de
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

JUGS e.V.
Java User Group Stuttgart e.V.
c/o Dr. Michael Paus
Schönaicherstraße 3, 70597 Stuttgart
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

Schweiz

JUGS
Java User Group Switzerland
Postfach 2322, 8033 Zürich
http://www.jugs.ch (info@jugs.ch)
Kontakt: Frau Ursula Burri

.Net User Groups

Deutschland

.NET User Group OWL
http://www.gedoplan.de/cms/gedoplan/ak/ms_net
c/o GEDOPLAN GmbH
Stieghorster Str. 60, 33605 Bielefeld

.Net User Group Bonn
.NET User Group "Bonn-to-Code.Net"
Langwartweg 101, 53129 Bonn
http://www.bonn-to-code.net (mail@bonn-to-code.net)
Kontakt: Herr Roland Weigelt

Dodned
.NET User Group Franken
http://www.dodned.de
Kontakt: Herr Bernd Hengelein
Herr Thomas Müller (consulting@tom-mue.de)

.net Usergroup Frankfurt
c/o Thomas Sohnrey, Agile IService
Mendelssohnstrasse 80, 60325 Frankfurt
http://www.dotnet-ug-frankfurt.de
Kontakt: Herr Thomas 'Teddy' Sohnrey
(thomas.sohnrey@gmx.de)

.NET DGH
.NET Developers Group Hannover
Landwehrstraße 85, 30519 Hannover
http://www.dotnet-hannover.de
Kontakt: Herr Friedhelm Drecktrah
(friedhelm@drecktrah.de)

INdotNET
Ingolstädter .NET Developers Group
http://www.indot.net
Kontakt: Herr Gregor Biswanger
(gregor.biswanger@web-enliven.de)

DNUG-Köln
DotNetUserGroup Köln
Goldammerweg 325, 50829 Köln
http://www.dnug-koeln.de
Kontakt: Herr Albert Weinert (info@der-albert.com)

.Net User Group Leipzig
Brockhausstraße 26, 04229 Leipzig
http://www.dotnet-leipzig.de
Kontakt: Herr Alexander Groß (agross@dotnet-leipzig.de)
Herr Torsten Weber (tweber@dotnet-leipzig.de)

.NET Developers Group München
http://www.munichdot.net
Kontakt: Hardy Erlinger (hardy.erlinger@hotmail.com)

.NET User Group Oldenburg
c/o Hilmar Bunjes und Yvette Teiken
Sachsenstr. 24, 26121 Oldenburg
http://www.dotnet-oldenburg.de
Kontakt: Herr Hilmar Bunjes
(hilmar.bunjes@dotnet-oldenburg.de)
Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

.NET User Group Paderborn
c/o Net at Work Netzwerksysteme GmbH,
Am Hoppenhof 32, 33104 Paderborn
http://www.dotnet-paderborn.de
(raacke@dotnet-paderborn.de)
Kontakt: Herr Mathias Raacke

.Net Developers Group Stuttgart
Tieto Deutschland GmbH
Mittlerer Pfad 2, 70499 Stuttgart
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
Kontakt: Frau Catrin Busley

.net Developer-Group Ulm
c/o artiso solutions GmbH
Oberer Wiesenweg 25, 89134 Blaustein
http://www.dotnet-ulm.de
Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

Österreich

.NET Usergroup Rheintal
c/o Computer Studio Kogoj
Josefgasse 11, 6800 Feldkirch
http://usergroups.at/blogs/dotnetusergrouprheintal/default.aspx
Kontakt: Herr Thomas Kogoj (thomas@kogoj.com)

.NET User Group Austria
c/o Global Knowledge Network GmbH,
Gutheil Schoder Gasse 7a, 1101 Wien
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx
Kontakt: Herr Christian Nagel (ug@christiannagel.com)

MATHEMA

meet the experts of enterprise infrastructure

Web-Anwendungen mit Apache Struts

Umsetzung von Web-Projekten mit Apache Struts
5. – 7. Dezember 2011, 14. – 16. März 2012
1.180,– € (zzgl. 19 % MwSt.)

Entwicklung für Android Smartphones

Mobile Anwendungen für Android
12. – 14. Dezember 2011, 18. – 20. Januar 2012,
1.180,– € (zzgl. 19 % MwSt.)

Persistenz mit Hibernate

OR-Mapping von Java-Objekten
11. – 13. Januar 2012, 3. – 5. April 2012
1.315,– € (zzgl. 19 % MwSt.)

Programmierung mit Java

Einführung in die Java-Technologie und die Programmiersprache Java
21. – 25. November 2011, 16. – 20. Januar 2012,
1.645,– € (zzgl. 19 % MwSt.)

Einführung in die Unified Modeling Language 2.x

Darstellung komplexer OO-Modelle mit der UML
25. – 26. Januar 2012,
2. – 3. April 2012,
835,– € (zzgl. 19 % MwSt.)

Lesen bildet. **Training macht fit.**

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

MATHEMA

Software-Entwickler (m/w)
Software-Architekt (m/w)

Arbeiten Sie gerne selbstständig, motiviert und im Team? Haben Sie gesunden Ehrgeiz und Lust, Verantwortung zu übernehmen?

Wir bieten Ihnen erstklassigen Wissensaustausch, ein tolles Umfeld, spannende Projekte in den unterschiedlichsten Branchen und Bereichen sowie herausfordernde Produktentwicklung.

Wenn Sie ihr Know-how gerne auch als Trainer oder Coach weitergeben möchten, Sie über Berufserfahrung mit verteilten Systemen verfügen und Ihnen Komponenten- und Objektorientierung im .Net- oder JEE-Umfeld vertraut sind, dann lernen Sie uns doch kennen.

Wir freuen uns auf Ihre Bewerbung!

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

Das Allerletzte

```
boolean isValid = obj == null ? false : true;
```

oder als Variation davon

```
boolean isValid = obj != null ? true : false;
```

wobei der Null-Check nur beispielhaft für einen booleschen Ausdruck steht.

Dies ist kein Scherz!
Es gibt wirklich Entwicklerfraktionen
die beide Konstrukte exzessiv verwenden.

Ist Ihnen auch schon einmal ein Exemplar dieser
Gattung über den Weg gelaufen?
Dann scheuen Sie sich bitte nicht, uns das mitzuteilen.

Der nächste KaffeeKlatsch erscheint Ende Dezember.

Herbstcampus

Wissenstransfer par excellence

3.– 6. September 2012

in Nürnberg